A flow measurement architecture to preserve application structure

A Flow Measurement Architecture to Preserve Application Structure

Myungjin Lee[a,1,*], Mohammad Hajjat[b], Ramana Rao Kompella[c], Sanjay Rao[b]

[a] School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom
[b] School of Electrical and Computer Engineering, Purdue University, 465 Northwestern Avenue, West Lafayette, IN 47907, United States
[c] Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47906, United States

Abstract

The Internet has significantly evolved in the number and variety of applications. Network operators need mechanisms to constantly monitor and study these applications. Modern routers employ passive measurement solution called Sampled NetFlow to collect basic statistics on a per-flow basis (for a small subset of flows), that could provide valuable information for application monitoring. Given modern applications routinely consist of several flows, potentially to many different destinations, only a few flows are sampled per application session using Sampled NetFlow. To address this issue, in this paper, we introduce *related sampling* that allows network operators to give a higher probability to flows that are part of the same application session. Given the lack of application semantics in the middle of the network, our architecture, RelSamp, treats flows that share the same source IP address as related. Our heuristic works well in practice as hosts typically run few applications at any given instant, as observed using a measurement study on real traces. In our evaluation using real traces, we show that RelSamp achieves 5-10x more flows per application session compared to Sampled NetFlow for the same effective number of sampled packets. We also show that behavioral and statistical classification approaches such as BLINC, SVM and C4.5 achieve up to 50% better classification accuracy compared to Sampled NetFlow, while not impairing existing management tasks such as volume estimation too much.

Keywords: NetFlow, related sampling, network management.

1. Introduction

The tremendous success of the Internet, and the fertile ground for innovation that it provides has spawned a diverse range of applications, with new applications continually and rapidly emerging and gaining in prominence. The last decade alone has seen rapid growth in popularity of peer-to-peer (p2p) systems (*e.g.*, BitTorrent, Skype, PPLive), online social networks (*e.g.*, Facebook), and more recently, cloud-based applications (*e.g.*, Salesforce [1], Google Apps [2]). While Internet traffic was predominantly dominated by the Web in the 1990's, p2p traffic accounted for over 60% of traffic around 2005, and more recently, video-based applications such as YouTube are gaining in popularity.

Concurrent with the growth of new applications and changes in popularity across applications, we are continually seeing shifts in characteristics and communication patterns of existing applications. For instance, the characteristics of Web traffic have significantly changed with the average size of Web objects increasing from 12 KBytes in 2000 to 68 KBytes in 2007 [3]. Further, p2p applications such as BitTorrent are being redesigned so that communication is localized within ISP networks, rather than crossing ISP boundaries [4, 5].

The emergence of new applications, and their rapidly changing characteristics require network operators to continuously measure and monitor traffic characteristics in their networks. These measurements allow operators to, for instance, potentially re-provision their networks, detect any new types of undesirable behavior within applications (*e.g.*, p2p system vulnerabilities [6, 7, 8, 9], attacks on Ajax-based web services [10]) and in general, prepare their networks better against any major application trends. Further, such traffic monitoring must ideally occur in a ubiquitous fashion as applications characteristics may differ significantly depending on the location [11].

Router-level measurement solutions such as NetFlow represent the most widely deployed and popular approach used for traffic monitoring today. The widely available nature of NetFlow across many modern routers makes it an ideal candidate for ubiquitous low-cost network monitoring. Unfortunately, however, routers employ *packet sampling* to scale to high line rates, that makes NetFlow ill-suited to monitor the new range of applications evolving in the Internet today. In particular:

- Emerging p2p and cloud-based applications are routinely composed of many different flows to potentially different servers/hosts that are often geographically distributed. With random packet sampling, only a small subset of flows, if any, are sampled for an application session that comprises many different flows. This makes it difficult to accurately characterize application behavior from sampled data.
- Several researchers have pointed out the inadequacies of simple port-based classification for emerging applications such

☆ Portions of this manuscript appeared in IEEE INFOCOM 2011.

* Corresponding author. Tel.: +44 (0)131 650 2713.
Email addresses: myungjin.lee@ed.ac.uk (Myungjin Lee), hajjat@purdue.edu (Mohammad Hajjat), kompella@cs.purdue.edu (Ramana Rao Kompella), sanjay@ecn.purdue.edu (Sanjay Rao)

[1] The work was performed when the author was at Purdue University.

as p2p [12, 13, 14]. While several alternate approaches based on statistical techniques, or host behavioral patterns have emerged [15, 13, 16, 17, 18, 14, 19], much of this work has dealt with unsampled data. The effectiveness of these techniques is likely to degrade with random packet sampling.

Motivated by these limitations of random packet sampling, in this paper, we propose the notion of *related sampling* based on the following key idea: *Once a flow is sampled, all flows that are part of the same application session, are sampled with high probability.* Applying related sampling means that either an application session is (almost) fully sampled, or not sampled at all. Behavioral classifiers benefit from the extra information (of flows that are related) and characterization can be all the more accurate.

We explore the potential of related sampling in the context of the RelSamp architecture. Ideally, flows corresponding to the same application session must be identified as related. However, since determining this is hard, RelSamp considers all flows that contain the same source IP address created within a given amount of time from each other as related. This heuristic is motivated from a measurement study on a 13-hour campus trace. The RelSamp architecture incorporates related sampling with the help of three stages of sampling. First, we use a host selection probability that controls which host (identified using the IP address) gets selected for subsequent packet selection. Once a host gets selected, packets are subject to a flow-selection probability that governs the probability with which a flow that contains the host as the source IP address is created. Finally, the last stage of packet sampling dictates the rate at which flow records are updated. Thus, RelSamp biases packet and flow selection in favor of hosts that are already admitted.

Because RelSamp selects hosts based on source IP address, it can recognize flows from different hosts behind NAT as if they are from a single host. Hence, biasing host selection as proposed can be difficult in an environment where NAT devices are heavily deployed. As such, not all types of networks can rely on our approach. Instead, we identify two key important networks—*enterprise and campus networks*—where accurate application behavior monitoring and classification is crucial. We believe that NAT issue is less concern in the networks; it is relatively easy for operators to locate a right deployment place for RelSamp for mitigating the NAT issue. In that sense, RelSamp can be most suitable for those networks. By the same token, we ignore home and core networks from consideration of deploying RelSamp.

Our study is built upon the networks where NAT boxes are less deployed or operators have a full control over managing them. Under this condition, the paper makes the following contributions:

- We introduce *related sampling* that allows flows that are part of the same application session to be sampled with higher probability. Our architecture allows selecting a large majority of flows from a given application session thus allowing scalable monitoring and characterization of new and emerging Internet applications.
- Using real traces, we extensively evaluate the efficacy of RelSamp. In our results, we observe that RelSamp is capable of obtaining 5-10x more flows per application session compared to sampled NetFlow and flow sampling [20] without significantly compromising the accuracy of aggregate packet count estimates (less than 12%).
- Using real packet traces with payload, we study the impact of RelSamp on traffic classification. Specifically, we show that the classification accuracy of BLINC, SVM and C4.5 increases by up to 50% in comparison with the flows output by sampled NetFlow for flows that are not easily classifiable using port numbers.

2. Measurement Model

Consider a router at the edge of a large-scale campus network or an enterprise network, typically referred to as a *gateway*. Our goal in this paper is to facilitate scalable monitoring of application traffic in order to characterize, study and monitor application behavior in a continuous fashion at such enterprise gateways. Such application monitoring is critical for enterprise network operators due to many reasons. First, operators need to prepare against any application trends that may potentially affect their network. For instance, analysts are already warning about new security vulnerabilities exposed by Ajax-based Web services [10]. Similarly, p2p applications have been recently shown to contain several vulnerabilities that can potentially be exploited to launch DDoS attacks [9, 6, 8]. Second, operators typically continuously plan for reprovisioning their networks based on evolving application trends. Understanding application trends, such as number of flows a given application generates, or the number of bytes transferred within a given flow, or burstiness of flows generated by modern applications (*e.g.*, Ajax-based Web applications generate several requests asynchronously within milliseconds [21]) is essential to conducting "what-if" analysis.

Typical enterprise gateway routers today operate at 10 Gbps capacity and these line rates are poised to increase further as technology scales. For example, the Internet gateway link at the Purdue university campus today is already of 10 Gbps capacity. A network operator interested in continuously monitoring these routers has currently two major options: The first option is to instrument an optical tap to split traffic and mirror it to a capture device. While the capture card itself is quite costly (currently a 10 Gbps capture card costs around $20-30K), a more serious problem is the volume of data one needs to collect (a 1 TB disk will be filled up in approximately 2 minutes). Thus, a second option, which most network operators use today, is to collect flow-level information using an inherent monitoring capability of routers today in the form of NetFlow. NetFlow provides flow records that summarize the traffic characteristics at the gateway. Because NetFlow does not easily scale to high line rates, operators use sampled NetFlow (*e.g.*, at 10 Gbps, typically a 1-in-1024 sampling rate is used).

Unfortunately, sampled NetFlow samples only a random fraction of packets, in turn, leading to a random fraction of flows. Modern applications however routinely contain several flows to potentially many different destinations; inferring application characteristics from a small fraction of sampled flow records is quite difficult. Our goal in this paper is therefore to develop a

scalable measurement architecture for ubiquitous and continuous measurement of application traffic that addresses this shortcoming associated with sampled NetFlow. Before we can describe our architecture in §3, we need to clearly define an abstract model of application traffic *from the perspective of a router.*

2.1. Capturing relatedness of flows

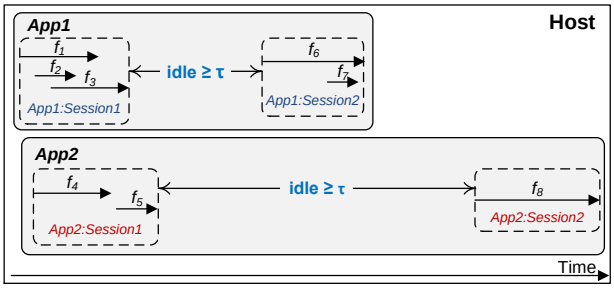
Typical measurement solutions operate at the granularity of a flow consisting of the 5-tuple (source and destination IP addresses, source and destination ports, and the protocol fields). However, application sessions typically involve many flows, potentially to many different destinations. A Web application session, for instance, consists of the set of flows a host originates in order to download Web objects from different Web servers. Thus, the fundamental unit of measurement in our framework is an *application session* that we define as follows:

Definition 2.1. *An application session is defined as the set of flows that correspond to a given application that originate at a given host (client, server or peer) to one or many other hosts (server, client, or peer) within a given amount of time of each other. To define more precisely, let us say that F_j is a set of flows of which the application type is j from the host. A flow is presented with a start time and an end time, and all flows in F_j are sorted in an increasing order of their start time. All the sessions of application j are then obtained by clustering the flows (starting from the first flow in F_j) with a maximum idle time of τ seconds. In other words, in an application session any flow is within a distance of τ seconds to at least one of its preceding flows.*

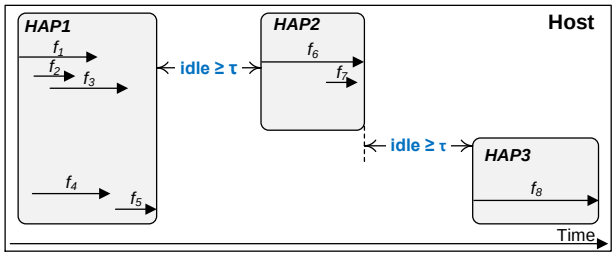
With our definition, only temporally clustered application activity is considered to be part of an application session. Since typical end users may use the same application at different points in time, our definition can help delineate between different instances of the same application. Further, our definition of an application session does not include transitive relationships because causality is hard to determine at a router in the middle of the network. For example, if a host A contacts a host B, which in turn contacts host C, then we consider them two different application sessions, one that originates at A and the other at B.

Figure 1(a) pictorially represents application sessions. The *x-axis* shows the time and each arrow represents a flow of a certain application type, originating from the same host. For example, App1:Session1 and App1:Session2 represent two sessions of the same application.

While application sessions provide a useful conceptual framework, it is difficult for routers in the middle of a network to identify flows belonging to the same application session. In recent years, it is no longer feasible to use port numbers to represent applications directly, as many applications routinely use non-standard ports [12, 13, 14]. Deep packet inspection could be used to identify and detect applications, however such techniques are too computationally expensive to be performed in an *online* fashion at routers on high-speed links [22]. While a few works [18, 23, 24] either focused on online classification or evaluated the classification speed of various algorithms. In particular,



(a) Application sessions



(b) HAPs

Figure 1: HAPs vs application sessions. $f_1 \ldots f_8$ represent flows generated by applications in a host.

a recent development [24] enables an online statistical classification for 10 Gbps link with a simple Bayes algorithm. However, it is yet unclear if all the existing algorithms (*e.g.*, SVM, C4.5, etc.) are suitable for real-time classification.

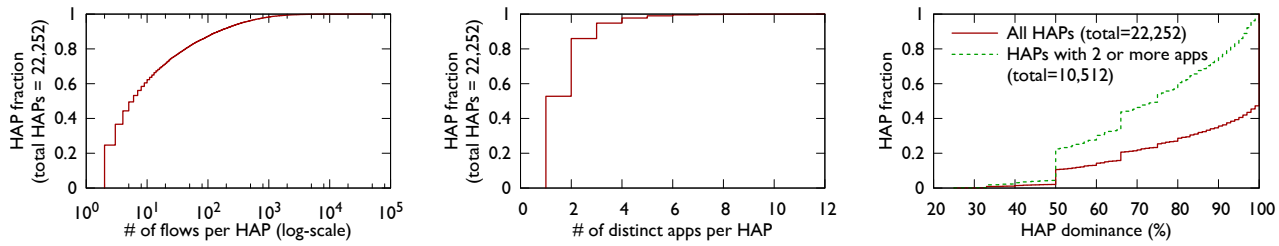
In order to deal with the scalability issues posed by online application classification in routers, we propose the notion of a *host activity period* (HAP) that is defined as follows.

Definition 2.2. *A host activity period (HAP) is defined as the set of flows that originate at a host (client, server or peer) to one or more destinations within a given amount of time of each other. To be specific, let us say that S is a set of all flows from the host, regardless of their application type, which are sorted in an increasing order of their start time. HAPs are obtained by clustering the flows (starting from the first flow in S) with a maximum idle time of τ seconds. In other words, in a HAP any flow is within a distance of τ seconds to at least one of its preceding flows.*

We observe that if a host runs exactly one application, then the corresponding application session and HAP are the same (assuming both definitions use the same threshold value τ). Comparing Figure 1(a) with Figure 1(b) shows this. All bursts of flows temporally close to each other originating from the same host are put together in the same HAP. Note that HAP1 encompasses two application sessions, App1:Session1 and App2:Session1, while HAP2 and HAP3 share a one-to-one correspondence with the actual application sessions.

2.2. HAPs vs application sessions

We now explore the relationship between HAPs and application sessions using an empirical measurement study. Our study is conducted using a 13-hour packet trace collected within a campus network (see §4 for details regarding the trace). The trace



(a) Number of flows per HAP (b) Number of applications per HAP (c) Dominance

Figure 2: Number of flows and applications per HAP.

contains full payload data, which enables us to identify the applications correctly using deep-packet inspection techniques (we use an open-source tool called `tstat` [25] for analysis). We use a τ value of 30 seconds for both the application session as well as HAP definitions for this study. Small values of τ could result in splitting application sessions (or HAPs) in a too fine-grained fashion, while large values could cause flows from many applications to be present in the same HAP. We have varied τ between 3 and 600 seconds, and found that the 30 second duration allowed us to have the least number of application sessions per one HAP without creating too many HAPs. We summarize our key findings:

• *HAPs can have a large number of flows:* Figure 2(a) shows the distribution of number of flows that constitute a HAP (for all HAPs that have at least two flows). As can be observed from the figure, only 22% of HAPs consist of two flows. Out of the remaining HAPs, more than 50% of them have at least 10 flows. A small percentage of HAPs (about 15%) consisted of greater than 100 flows (with a maximum around 75,000 flows) indicating that host activity can be quite intensive.

• *Most HAPs have a small number of applications:* Figure 2(b) shows the distribution of number of applications per HAP. We can observe that a majority of HAPs consist of a very small number of applications, with almost 50% of HAPs consisting of only one application. For such cases, HAPs are identical to the application sessions. About 85% of HAPs consist of less than two applications. Some HAPs consist of all the way up to 12 applications. We suspect this is because of applications that run in the background within a host that constantly generate flows. A HAP is delineated only if there is a gap of greater than τ between two flows, which may not have occurred in these cases because of the background activity.

• *For HAPs with more than one application, there typically exists one dominant application.* Finally, among those HAPs which have more than one application, we plot the dominance of HAP in characterizing an application session in Figure 2(c). We define the dominance of a HAP as the percentage of flows that belong to the application that has the most flows in the HAP, i.e., m/t where m is the maximum number of flows among all applications and t is the total number of flows within the HAP. We observe from Figure 2(c) that more than 80% of the HAPs have a dominance of greater than 70%. Of course, this includes those HAPs trivially which have only one application for which the dominance is 100%. Thus, if we discard those single application HAPs that contain only one application, we can still observe that more than 50% of the HAPs have a dominance of greater than 70%.

Implications for RelSamp **design:** Our results based on this measurement study indicate that the notion of HAP captures application sessions quite effectively. Further, given that HAPs (application sessions) can have a large number of flows, the results indicate the importance of capturing related flows to get representative characterizations of applications. Note that the existence of a few instances when a HAP may contain many different application sessions, does not pose explicit problems in our measurement framework. This is since, we can post-process the collected flow records within a HAP by applying classification approaches such as BLINC [14] to split a HAP into multiple application sessions. Given these results, our focus in designing RelSamp is to capture as many flows per HAP as possible.

3. Architecture of RelSamp

Sampled NetFlow is the most commonly used solution for monitoring flows in the Internet. Sampled NetFlow works by sampling each packet with a pre-configured sampling probability (*e.g.*, 1 in 1024). For each sampled packet, if a flow record already exists for the flow (identified by the 5-tuple) that the packet belongs to in the flow cache, the flow record is updated (*e.g.*, packet and byte counters are incremented). Otherwise, a new flow record is created with the flow key of the packet. Unfortunately, because of its random packet selection process, it captures very few flows per HAP, which our RelSamp architecture attempts to solve.

3.1. Design

The *key idea* of RelSamp is to create flow records by sampling flows that are related to already sampled flows with higher probability so that more flows within a HAP are collected. This small bias, as we shall show later in our evaluation, is remarkably effective at collecting many more number of flows per application than with Sampled NetFlow, thus facilitating better application classification as well as continuous and ubiquitous characterization of application traffic.

The basic design of RelSamp is shown in Figure 3, and is very similar to NetFlow in terms of the flow memory and the flow

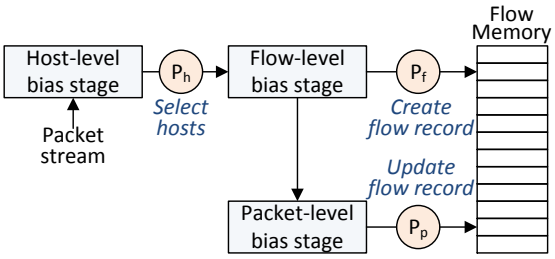


Figure 3: Architecture of RelSamp.

record structure. It mainly differs from Sampled NetFlow in its sampling mechanism. Conceptually, RelSamp consists of three stages of sampling—host selection, flow creation and flow update. The purpose of the host selection stage is to select a HAP for measurement. Once a HAP is probabilistically selected, the flow selection stage determines what percentage of the flows within the HAP to observe. Flows for which flow records exist are then updated periodically with a flow update probability. The accuracy of flow statistics for each created flow record is governed by the flow update probability. Algorithm 1 presents the pseudocode for the entire RelSamp packet sampling algorithm. We now describe the individual stages in more detail.

Host sampling. There are many different architectural options for sampling HAPs. One option is to maintain some amount of state for active HAPs in the form of a hash-table (call it HAP table). If the source IP address of a packet is not already contained in the HAP table then, the host could be sampled with some probability, p_h and an entry could be created in the HAP table. The advantage of this approach is that, only HAPs with sufficient volume will be sampled. Unfortunately, the above option has two problems. First, it requires extra state maintenance to create and expire HAP table entries, that may be complicated. Second, some number of packets would be missed before flow records are created, that need to be accounted for. These packets may belong to several flows, and it is not clear how to create unbiased estimators for flow records.

Instead, in RelSamp, we sample HAPs using a *hash-based selection* on IP addresses similar to flow-sampling in [20]. In other words, we focus only on a subset of source IP addresses that are selected by hashing these source IP addresses[2] and checking whether the hash lies within a pre-configured range (see line 7 in Algorithm 1). For instance, let us assume that hash value 0x10 is obtained by hashing IP address 1.0.0.1, p_h is set to 0.1 and U is 0xFF. Because 0x10 is less than $p_h \cdot U$ (*i.e.*, 0x19), the host IP is selected. Thus, flows that clear this stage are likely to belong to the same application (thus they are related and will be further considered for sampling in subsequent stages). By controlling the hash range to the total hash space, we can control the host sampling rate. Because hosts are either selected on the first packet or not at all, no packets are missed before a host is selected at

Algorithm 1 Packet selection process

```
1:  procedure SELECT-PACKET(p_h, p_f, p_p, P)
2:      ▷ U: a maximum number presented by a hash space
3:      ▷ r_f ∈ [0,1): a random no. for flow selection
4:      ▷ r_p ∈ [0,1): a random no. for packet selection
5:      ▷ F: flow memory
6:      (srcIP, srcPort, dstIP, dstPort, Proto) ← key(P)
7:      if hash(srcIP) > p_h · U then
8:          return
9:      if r_f > p_f and r_p > p_p then
10:         return
11:     if key(P) ∉ F then
12:         if r_f ≤ p_f then
13:             create_flow(F, P)
14:     else if r_p ≤ p_p then
15:         update_flow(F, P)
```

the flow-level. Thus, unbiased estimators are easy to create in this framework, as we shall describe in §3.2. Further, it does not require any additional HAP table state for maintaining HAP entries. HAPs could be easily constructed by post-processing the sampled flow records that contain the start and end timestamps anyway (by checking whether the flows are separated by more than τ). A potential concern with hash-based selection is that, only certain hosts will be selected, while others may not be sampled at all. We can easily solve this by changing the hash function periodically[3]. Alternately, we can choose higher sampling probabilities for p_h to minimize its impact.

Flow creation. Once a particular host is selected based on the hash-based host selection, the packet is handed to the next stage where a flow is created for the packet, if it does not exist already, with a probability p_f. This stage presents network operators with the flexibility to choose what percentage of flows for a given host are selected. At this stage, again one can choose either hash-based or packet-based selection. Packet based selection creates flow records for heavy-hitter flows, while hash-based selection will create flow records for all types of flows. We pick packet-based selection since volume estimates are more accurate.

Flow update. The final stage in our RelSamp architecture is the flow update stage. Packets for flows that are already existing in the flow cache are updated with probability p_p. This gives an operator additional flexibility to specify the accuracy with which individual flow records are updated. In Sampled NetFlow, the flow creation and the flow update probabilities, p_f and p_p are equal to the configured sampling probability. The reason we split this base probability into two parts is to provide network operators the ability to trade-off accuracy of each flow with more number of flows per host. Thus, for a given *effective sampling rate* p_e (that dictates how many effective packets are sampled), we can choose to provide a higher p_f that allows creating more

[2] At a router, traffic destined to both clients and servers is observable. Thus, using dstIP may be able to effectively collect related flows. Another possibility is to selectively use srcIP or dstIP based on the origin of flows. Although not explored yet, these methods may marginally improve the performance of capturing related flows because our method already works well (see §5.1).

[3] This can be also useful to make our mechanism robust against adversarial traffic patterns such as DDoS attack. For instance, if an attacker somehow knows the hash function used in RelSamp, he can make all the attack flows always pass the host selection stage by carefully assigning source IP addresses to bot machines. Changing hash functions regularly can mitigate this kind of issue.

number of flows, each of which is updated with a lower sampling probability p_p.

For the purposes of increasing related application flows, which is the main goal of our architecture, we need to increase the number of flows that share common source IP address (by setting p_f to be high). Of course, we still need to ensure that the total number of packets lies within a given packet sampling budget, p_e, which naturally requires configuring the value of either p_p or p_h to be small. The effect of reducing the p_p value is that, individual flow statistics may suffer from higher errors. The effect of reducing the p_h value is that aggregate volume estimates become more inaccurate, as scaling the volumes from the observed hosts to the actual hosts becomes skewed as number of hosts decreases. Thus, there is a natural trade-off between these three variables, that need to be configured according to the objectives and dependent on the particular location at which sampling is being performed. We demonstrate how to set up the three parameters in §4.

3.2. Unbiased Estimators

We now show how to construct unbiased estimators for packet and byte counts per-flow. Unbiased estimators are critical mainly for volume estimation tasks without which errors can be really high, especially for aggregates constructed from individual flow records. We begin our discussion by providing an estimator of per-flow packet counts. Note that since host selection process is hash-based, no packets are lost before selecting a host; thus, the packet count estimate is dependent only on the p_f and p_p probabilities.

Estimation of per-flow packet counts. Let s be the actual number of packets for a flow f and c be the total number of packets sampled in the counter. The unbiased estimator $\widehat{s}$ for the number of packets is given by the following equation.

$$\widehat{s} = \frac{1}{p_f} + \frac{c-1}{p_p} \tag{1}$$

Proof: Intuitively, packet selection process for a given flow packet counter can be thought of as a sequence of rounds, with each round involving a sequence of unsampled packets finally terminating with a sampled packet. The final value of the counter c indicates the total number of rounds. Let us suppose $\widehat{s_i}$ be the random variable indicating the set of packets comprising the round i. Assuming packets are sampled with probability p_i within the ith round. The unbiased estimator $\widehat{s_i}$ is given by $\widehat{s_i} = 1/p_i$, which is the standard unbiased estimator for a geometric random variable.

As the packet sequence length is the sum of the lengths of individual rounds, $\widehat{s} = \sum_{k=1}^{c} \widehat{s_i}$. The probability for the first round $p_1 = p_f$, and for all other rounds, $p_j = p_p$, $1 < j \leq c$. Also, these individual random variables are independent of each other. Thus, the unbiased estimator for the packet count for a flow can be computed as follows.

$$\widehat{s} = \frac{1}{p_f} + \sum_{k=1}^{c-1} \frac{1}{p_p} = \frac{1}{p_f} + \frac{c-1}{p_p} \tag{2}$$

■

Variance estimate. The variance of this estimator can be computed similarly as follows. First we compute the variance of the individual s_is as follows.

$$Var[\widehat{s_i}] = \frac{1-p_i}{p_i^2} \tag{3}$$

The above expression is the variance estimate of a standard geometric random variable. Similar to the mean estimates, we can sum the individual variances to obtain the total variance of the estimate.

$$\begin{aligned} Var[\widehat{s}] &= Var[\sum_{k=1}^{c} \widehat{s_i}] = \sum_{k=1}^{c} Var[\widehat{s_i}] \\ &= \frac{1-p_f}{p_f^2} + (c-1)\frac{1-p_p}{p_p^2} \end{aligned} \tag{4}$$

Estimation of per-flow byte counts. Let b_i ($1 \leq i < l$) represent the byte size of individual packets for a given flow. The total byte count of a flow is represented by $b = \sum_{i=1}^{l} b_i$. Let S_c be the set of indices of sampled packets with probability, p_p, and c ($1 \leq c \leq l$) denotes the cardinality of the set, ($|S_c|$). Let, b_{first} represent the size of the first packet sampled. The following equation is an unbiased estimator $\widehat{b}$ of per-flow byte count b.

$$\widehat{B} = \frac{b_{first}}{p_f} + \sum_{i \in S_c} \frac{b_i}{p_p} \tag{5}$$

We can prove that this estimator is unbiased in a similar fashion to the packet count estimator. Essentially, the byte count estimators for each of the s_i sequence of packets is given by $\widehat{B_i} = b_i/p_i$, where b_i is the byte count of the sampled packet in round i. We can compute the estimate for byte count $\widehat{B}$ trivially by adding up the individual $\widehat{B_i}$s. Variance of the byte count estimate, on the other hand, is hard to compute this way, unless we assume packet sizes are distributed uniformly. This assumption is not true in general; we have not yet been able to compute a general formula for estimating this and consider it part of future work.

4. Experimental Setup

In this section, we first lay out main criteria to evaluate our architecture followed by a brief discussion about sampling methods including RelSamp. We then explain the datasets that we collected for the experiments.

Evaluation objectives. We answer the following questions throughout the next two sections (§5 and §6):

1. How effective is RelSamp in sampling related flows that belong to an application session?
2. Are the estimators in RelSamp unbiased ? What is the relative inaccuracy in estimating flow volumes ?
3. What are the limitations of using random packet sampling in enabling applications such as traffic classification?
4. How effective is RelSamp in ensuring better classification accuracy as compared to random packet sampling?

Sampling methods. We implement three sampling methods—flow sampling, sampled NetFlow, and RelSamp by extending an open-source NetFlow called YAF [26]. In flow sampling, a flow sampling rate is determined by controlling a hash range to the the total hash space. A packet is hashed using a 5-tuple flow key, and if the hash value falls in the hash range, the packet is then selected. In this manner, the flow sampling can consistently update all the packets of selected flows. On the other hand, random packet sampling employed by NetFlow selects packets uniformly with a specified sampling probability. We implement this method by generating a random number between 0 and 1 and selecting the packet if the random number is less than the specified probability. RelSamp is implemented in the same way described in §3. Note that RelSamp and flow sampling are different in that our scheme has a host bias stage while flow sampling does not. In addition, our scheme can control the number of packets that are sampled in a flow whereas flow sampling shows all-or-nothing nature in the process. Thus, as we will show shortly, because of these differences, flow sampling should capture even more number of packets than our scheme so that it can obtain the same number flows per HAP as our scheme does.

Datasets. We make use of two datasets. The first dataset, *CAMPUS*, is an OC-192 (10 Gbps) packet-header trace collected at the edge of a large university campus. The trace is an hour long, and consists of about 140 million flows and 1,293 million packets. The purpose of using this trace is to answer questions about how to choose the parameters of RelSamp in edge network settings.

Our second dataset, *DORM*, is a packet trace with full payloads collected from a router on a large dormitory building in the campus. A full-payload trace allows us to evaluate the implications of RelSamp for traffic classification by enabling deep packet inspection (DPI) techniques to establish ground truth, i.e., determine the actual application (see §6). The trace is about 13 hours long. We filter out traffic local to the university, and the resulting trace consists of around 214 million packets distributed over 8.5 million flows and carrying around 139 GBytes of volume. Since we had access to the packet payload from DORM trace only and not from the CAMPUS trace, we use this trace to obtain application ground truth in order to evaluate the classification accuracy using BLINC and statistical methods. Note that flows are different as long as the typical 5-tuple (*i.e.*, srcIP, srcPort, dstIP, dstPort, protocol) flow keys of flows in comparison are not same. Thus, even if the flow keys of two flows can be identical by reordering them (because their directions are opposite), they are two separate flows (*i.e.*, two uni-directional flow records) regardless of sampling techniques used in our work. That is, sampled flow records represent uni-directional flows that originate at either clients or servers.

Setting parameters of RelSamp. We discuss how the parameters of RelSamp should be chosen so as to enable the most desirable sampling schemes. We begin by discussing the impact of each of the parameters. Recall that the host selection probability p_h controls the total number of hosts, and thus the total number of packets, directly impacting the aggregate volume estimation accuracy. On the other hand, flow selection probability p_f governs application awareness, while packet selection probability p_p dictates the accuracy of per-flow statistics. Thus, there exists a clear tradeoff in configuring these three parameters depending on the needs of the network operator, measurement location, link capacity, and router resource constraints.

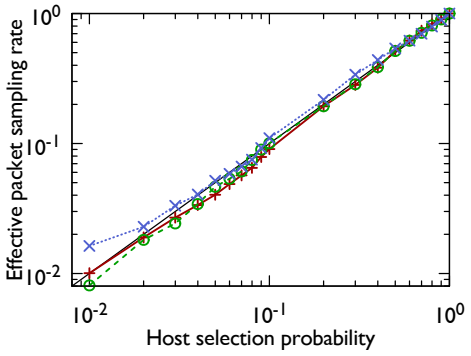


Figure 4: Relationship among host selection probability p_h and effective packet sampling rate p_e for different random runs. CAMPUS trace is used in this experiment. For the setting, while both p_f and p_p are fixed as 1.0, the parameter p_h is varied from 0.01 to 1.0.

We first investigate the impact of the host selection probability p_h. Theoretically speaking, as we increase the host sampling probability, we expect that the total number of packets should increase roughly linearly. Figure 4 plots the effective sampling rate p_e as a function of p_h for the *CAMPUS* trace for different random runs, while fixing $p_f = p_p = 1.0$. Here, the effective sampling rate p_e is just the fraction of packets sampled. As expected, p_e increases almost linearly with the host selection probability p_h. However, packets are slightly over- or under-sampled (for different runs) when $p_h \leq 0.02$ because as the p_h value reduces, the number of effective hosts considered reduces causing the sampled hosts' traffic profile to deviate farther away from the overall population. Although not a strict requirement, it is desirable not to set p_h value too low (~ 0.03 for this trace) so that the sampled traffic volume is not significantly higher or lower than the desired sample size ($p_h \times T$, where T is the total traffic), and the application characteristics of sufficient number of unique hosts are factored in the measurements.

Given p_h, configuring p_f and p_p can be slightly tricky. The complexity comes from the fact that flow size distribution is non-uniform. Thus, a slight increase in p_f may require a large decrease in p_p to meet a given effective sampling rate p_e. Our investigation with the CAMPUS trace revealed this non-linear relationship between p_f and p_p. We mitigate this issue by empirically calibrating these two parameters.

Given p_e, we first choose p_h ($p_h >= p_e$) and a target flow sampling probability p_f^t depending on monitoring purposes. Then, we set both p_f and p_p as p_e/p_h, which guarantees an effective sampling probability of p_e. Our scheme iteratively does binary search for p_f and p_p over the incoming stream until we get a configuration where p_f gets close to p_f^t without violating the budget of p_e. Until we obtain a stable set of the three parameters, intermediate flow records are discarded.

We sketch this search scheme as follows: p_f is set to $(p_f + p_f^t)/2$ and p_p to $(p_p + \delta)/2$, (δ ($= 10^{-5}$) is a minimum sam-

Setting	p_h	p_f	p_p
RelSamp1	0.03	0.76	0.0001
RelSamp2	0.005	1.0	0.3

Table 1: Different settings of RelSamp probabilities optimized for different objectives while keeping the effective sampling rate the same at $p_e = 0.001$.

pling probability that can be set) and the achieved packet sampling rate p_a is tested. If p_a is larger than p_e, p_p is further reduced by setting it as $(p_p + \delta)/2$. If p_a is smaller than p_e, p_p increases by setting it as $(p_p + 1.0)/2$. Once p_a becomes close to p_e, we stop adjusting p_p and continue to calibrate p_f by setting p_f as $(p_f + p_f^t)/2$. As a consequence, this may result in readjustment of p_p. We obtain two settings shown in Table 1 using this heuristic. Note that the heuristic may cause temporal instability in terms of accuracy and overhead against the sampling budget, but once a setting settles down, further calibration is unnecessary for long term monitoring tasks.

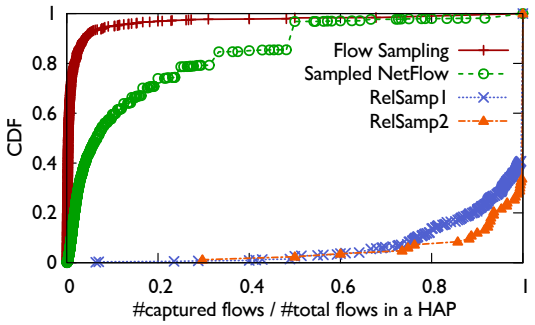
5. Basic Evaluation

We in this section answer the first two questions presented in §4. In particular, we investigate how effectively RelSamp captures related flows that belong to an application session, and evaluate its accuracy in estimating flow volumes.

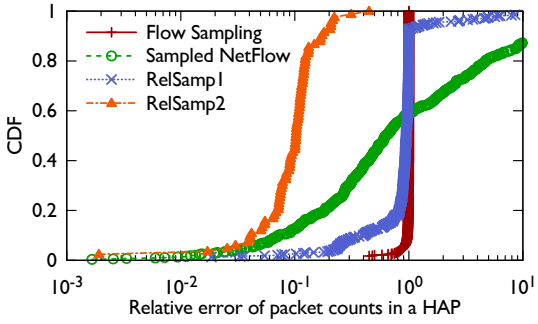
5.1. *Sampling related flows*

We explore the effectiveness of RelSamp in sampling flows corresponding to the same HAP, and the sensitivity of the results to the parameters of RelSamp using the two configurations shown in Table 1. The two settings result in similar effective sampling rate of 0.001, but have different proportion of p_h and p_p. Higher p_h essentially chooses higher number of hosts for consideration, while lower p_p results in slightly higher error for individual flows. Depending on the importance of different objectives, one could choose different settings. For comparison, we use two other sampling schemes, sampled NetFlow with 0.001 sampling rate and flow sampling (with probability such that 0.001 fraction of packets are sampled).

Our primary evaluation metrics are *flow coverage*, which we define as the fraction of the flows captured by Sampled NetFlow or RelSamp per HAP, and *volume estimate* defined as the total number of packets per HAP captured by these algorithms. Figure 5(a) plots the CDF of the flow coverage obtained across the HAPs. Only HAPs with at least two flows are considered; considering HAPs with one flow will shift the curves slightly to the left. Each curve corresponds to results with a particular sampling algorithm (NetFlow, as well as two settings of RelSamp). From the figure, we can make several observations. First, Flow Sampling (topmost curve) is the least effective in ensuring flow coverage. Second, NetFlow is slightly better than Flow Sampling—only 10% of the HAPs see a flow coverage of 50% or more, with a median flow coverage of only 10%. Third, as we move from RelSamp1 to RelSamp2, the flow coverage curve moves to the right. We can see that RelSamp2 performs better than RelSamp1 due to the fact that p_f is higher in RelSamp2 setting than RelSamp1.



(a) Flow coverage

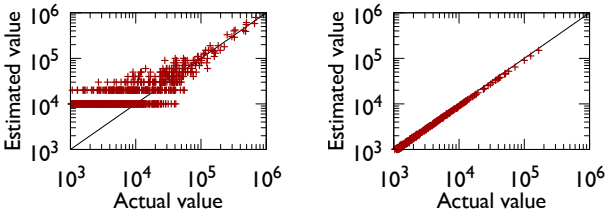


(b) HAP volume error

Figure 5: Flow coverage and volume estimate of a HAP. We use $\tau = 30s$ and use the CAMPUS trace for this experiment.

Our most aggressive setting RelSamp2 achieves more than 90% flow coverage for over 90% of flows. Our RelSamp1 and RelSamp2 curves clearly indicate almost **5-10x** increase in the median flow coverage compared to sampled NetFlow and almost **10x** increase compared to the flow sampling curve, which performs the worst in all the experiments due to its inability to preserve any application semantics. As an aside, we note that with the RelSamp2 setting, even though $p_f = 1.0$, the flow coverage can be observed to be less than 1 in Figure 5(a). This is because of NetFlow's (in)active timeouts and multiple normal flows with the same flow key that makes it difficult to match the NetFlow's output with HAPs exactly.

In Figure 5(b), we compare the relative error in estimating a HAP's volume for different sampling schemes, *i.e.*, we consider the error in estimating the total volume across all flows that constitute a HAP compared to the ground truth. We can observe once again that flow sampling curve is significantly worse than the rest, with a relative error of almost 100% for more than 95% of HAPs. Sampled NetFlow performs slightly better compared to flow sampling, but still, the median error is close to 100% error. RelSamp1 performs better than both flow sampling and Sampled NetFlow, but because the individual flow volume estimates are not that accurate, it also suffers from worse error. RelSamp2 performs the best since it is the most aligned with our goal, namely, higher flow coverage and accurate flow volume estimates ensured by the higher value of p_p, which is set at 30%. This phenomenon is also shown in the scatter plots we draw to show unbiasedness of our estimators in the next subsection.



(a) RelSamp1 (b) RelSamp2

Figure 6: Scatter plot of actual and estimated packet counts using RelSamp1 and RelSamp2 settings.

Actually, one could argue that RelSamp2 is actually *strictly* better than RelSamp1 since it seems to outperform RelSamp1 in achieving less relative error of the HAP volume estimates as well as in capturing more number of flows per HAP. While this is true, the advantage of RelSamp1 compared to RelSamp2 is in using a larger fraction of hosts than RelSamp2. This advantage may matter when one considers other application characteristics where such a difference is important. Overall, our architecture provides the flexibility to choose arbitrary settings depending on the goals of a network operator.

5.2. *Volume estimation*

Unbiasedness of estimators. We empirically validate our unbiased estimators next. Figure 6(a) first shows the scatter plot of actual and estimated packet counts for flows containing at least 1,000 packets using the RelSamp1 setting. The two-sided errors from the figure indicate the unbiased nature of our estimator[4]. As flow size increases, the actual and estimated flow sizes converge and the relative error becomes smaller. We observe a similar trend for the scatter plot corresponding to the RelSamp2 setting as shown in Figure 6(b), except that the estimates are much more accurate than that of RelSamp1. The reason for this is quite obvious; the packet sampling probability p_p under the RelSamp2 setting is quite high (30%) compared to the (0.01%) setting in RelSamp1. This allows RelSamp2 to be more accurate in the individual flow volume estimates.

Per-application volume estimation. We evaluate whether RelSamp preserves the relative volumes of applications. We aggregate flow records based on port numbers (for a few popular applications)—that is, when either port number (source or destination) matches the port number of interest—and show the histograms for the relative percentage estimates of both packet and byte counts for each of these aggregates in Figure 7. We show histograms for the true volume, Sampled NetFlow and the two settings of RelSamp as described in Table 1.

[4]Note that in RelSamp1 setting, the p_p is set 0.01%. Thus, for flows that have less than 10,000 packets, our estimator tends to overestimate their sizes. On the other hand, for flows with more than 10,000 packets, we observe that the number of overestimated samples are roughly equal to that of underestimated samples. This kind of inaccuracy even occurs in case of the unbiased estimator of simple random sampling when flow sizes are less than $1/p$ where p is random sampling probability

Flow size	NetFlow		RelSamp1		RelSamp2	
	pkt	byte	pkt	byte	pkt	byte
$> 10^{-2}$%	1.43%	0.94%	1.87%	2.06%	0.22%	4.85%
10^{-3}-10^{-2}%	0.55%	0.84%	7.25%	16.89%	10.00%	2.00%
10^{-4}-10^{-3}%	0.26%	0.71%	11.52%	7.74%	9.18%	6.63%

Table 2: Relative error of volume estimates by aggregating flows by flow size.

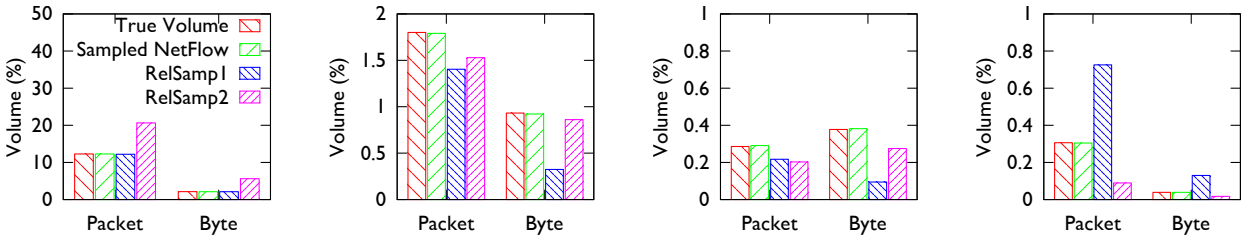
We observe the following from Figure 7. First, we can see that the estimates of per-application volumes are reasonably close to the true value. The discrepancy is slightly more pronounced for low volume applications, because they contain far fewer flows and if the right hosts are not sampled, they are likely to suffer from under- or over-estimation. Second, as expected, RelSamp is slightly less accurate than Sampled NetFlow, but this is the price RelSamp pays for the more than 5-10x increase in the median flow coverage as shown in Figure 5(a).

Aggregate volume estimation. We also consider a different form of aggregation, based on the flow sizes. We consider three different groups of flows similar to [27], that have a volume of > 0.01% (large flows), between 0.001% to 0.01% (medium flows) and finally, 0.0001% to 0.001% (small flows) as shown in Table 2. We compute our results on the hash space and re-normalize according to the empirically determined re-normalization factor we described in §3. Again, as expected, the relative error of Sampled NetFlow is very low compared to different instances of RelSamp that have different degrees of inaccuracies depending on the particular choice of parameters. Overall, the errors are still within 10% for RelSamp2, and within 17% for RelSamp1 despite the clear adversarial setting for packet count estimates.

6. Impact on Traffic Classification

Network operators need to classify traffic to enable services such as traffic differentiation and estimating volumes of individual applications. Due to the increasing inadequacy of port-number classification— p2p applications (*e.g.*, BitTorrent) routinely use non-standard ports, emerging applications (*e.g.*, Ajax-based Web services [2], embedded video [28]) all run on port 80—it is important to use more sophisticated classification techniques. While researchers have proposed several approaches based on host-behavior [14, 19] and machine-learning techniques [15, 13, 16, 17] for classification, pretty much all of this work assumes unsampled data. Our focus, in contrast, is to study the impact of sampling on traffic classification.

In this section, we shed light on (i) the effect of sampling on traffic classification techniques; and (ii) the effectiveness of RelSamp in aiding traffic classification when compared to NetFlow and Flow Sampling [20]. In our study, we analyze BLINC, a host-behavior based classifier, and two machine-learning algorithms: C4.5 and Support Vector Machine (SVM). We start by a brief description of each, followed by the methodology used to evaluate them, and end with detailed results.



(a) HTTP(80) (b) HTTPS(443) (c) SMTP(25) (d) DNS(53)

Figure 7: Comparison of relative volume estimates for different ports (of popular application types). Note that this port-based grouping does not serve for classification purpose; it is merely used for convenience to depict how accurate RelSamp's volume estimation can be depending on port numbers.

6.1. *Approaches to traffic classification*

Two key approaches have emerged for traffic classification:

Host-behavior based classification. This approach shifts the focus from classifying individual flows to associating hosts with applications, and then classifying their flows accordingly [14, 29]. The intuition behind it is that observing the activity and social interactions of the hosts provides more information and can reveal the nature of the applications of the host. BLINC [14, 29], one of the most well-known classifiers, captures host profiles by looking into each host's interactions with other hosts. The interactions of a certain application is captured through an empirically derived signature, called a *graphlet*. A graphlet reflects the "most common" behavior for a particular application. It captures the relationship between the use of source and destination ports, the relative cardinality of the sets of unique destination ports and IPs as well as the magnitude of these sets. A host is classified by identifying the closest matching behavior in the built-in library of graphlets. More details can be found in [14].

Supervised machine learning techniques. These classifiers require training with data that is labeled in advance with the ground truth. The learning algorithm has to generalize using the presented training data over unseen situations in the testing data. We test the effect of our sampling scheme on two well-known algorithms: *C4.5* and *SVM*. The C4.5 algorithm [29] builds a decision tree based on training data, which can then be used for traffic classification. SVM [30, 29] constructs a hyperplane or set of hyperplanes in a high-dimensional space which can be used for traffic classification. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class. SVM is proposed as a robust classifier in [29] which consistently outperformed all other classifiers tested across all traces.

6.2. *Methodology*

Our evaluations are conducted using packet traces collected with full packet payload required to establish the ground truth. Figure 8 illustrates the dataflow used to evaluate classification techniques. Four different methodologies are parallel applied to the packet-level trace: a DPI-based classifier (to obtain ground truth), RelSamp, Sampled NetFlow, and Flow Sampling. There have been several approaches/tools [31, 32, 25] that obtain the ground truth on flows' application types. Among others, we use

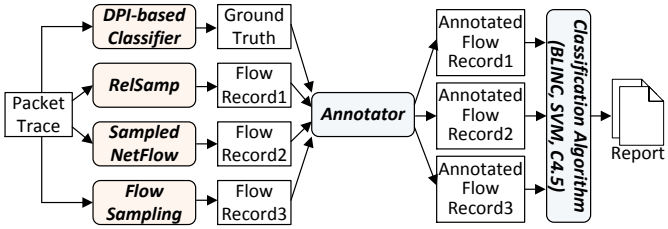


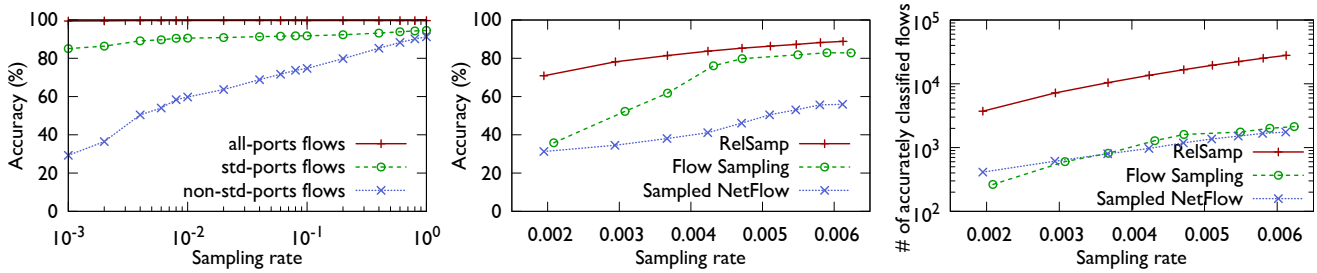
Figure 8: Dataflow used for evaluation of classification techniques.

a deep packet inspection (DPI) tool called `tstat` [25] and annotate raw unsampled flows with application types. `tstat` provides ground truth by annotating each flow with its application type after parsing its content searching for signatures of applications it can recognize. In our evaluations, we ignore flows for which `tstat` is unable to determine their application type (*e.g.*, encrypted flows). In our trace, less than 8% of the total flows were unable to be classified by `tstat`. The flows generated by RelSamp, Sampled NetFlow and Flow Sampling are annotated by correlating them with the unsampled flows. Sampled flows are then fed to the different classification techniques.

Traffic flows conforming to well-known standard ports are easy to classify using a simple classifier that takes port number information for classification. However, classification of flows that do not use their well-known standard ports is far more challenging. We therefore dissect traffic into three categories depending on their use of standard ports: (i) *std-ports flows* which are flows that use any of the standard well-known ports for that flow type (*e.g.*, port 80 for Web, 110 for POP3, and 443 for SSL)[5]; (ii) *non-std-ports flows* which are flows that do not use the standard known port for the flow's application type; and (iii) *all-ports flows* which include all flows independent of their ports. In the DORM trace, we found that 42% of the total flows were *std-ports flows* and the remaining 58% of the flows were of the *non-std-ports flows* type. In the rest of the analysis, we mainly focus on the second category.

We employ *Reverse BLINC* [29], an extension of original BLINC, with the default values of configuration parameters. Reverse BLINC overcomes the limitation of misclassification of

[5]Note that this categorization of flows is not about classification. For instance, some flows using port 80 are *std-ports flows* but their application type may not be Web; the ground truth is obtained from `tstat`.



(a) Accuracy for different categories of flows generated by Sampled NetFlow

(b) Accuracy for *non-std-ports flows*

(c) Number of accurately classified flows for *non-std-ports flows*

Figure 9: BLINC's classification accuracy results.

non-bidirectional traffic which existed in the original BLINC version. For machine learning classification, we use a well-known data mining software suite called WEKA [33]. We use Sequential Minimal Optimization (SMO) [34], a well known algorithm for training SVM, and used parameter settings that have been shown to work well in prior work [17, 29]. For the machine learning algorithms, we use the following flow features: *protocol, source and destination port numbers, average packet size, and TCP flags: RST, ACK, SYN, and FIN* (these flags are set to zero for UDP flows). These features have been shown to work well [29]. We note that [29] derived this subset from an exhaustive list of features using the Correlation-based Filter (CFS) algorithm [35] to filter out irrelevant and redundant features.

We test classification techniques using one hour of packet data from the DORM trace. The same trace is used as input for all classification techniques. Supervised machine learning techniques also require training so we use another hour from the same trace to train both SVM and C4.5. For each sampling algorithm and sampling rate, we train and test SVM and C4.5 with data sampled using those settings. For example, if we are to test SVM on traffic sampled with rate 0.001, the same sampling rate of 0.001 is also used for training. We take this approach to ensure features are used in a consistent fashion during training and testing, which in turn, can ensure accuracy of the classifiers. Consider a feature such as the number of packets in a flow, for instance. Clearly, the values obtained for this feature are impacted under sampling. Training and testing with the same sampling rate can ensure consistent use of the feature.

For the parameter setting of RelSamp, while we fix $p_h = 0.2$ and $p_p = 0.0001$, we varied p_f from 0.1 to 0.9. We set parameter values which are higher than ones in Table 1 because host population and flow sizes in DORM trace are smaller than those in CAMPUS trace. In addition, because we wish to investigate the influence of p_f on traffic classification, we do not fix any given p_e, but let it vary freely by varying p_f values. Thus, p_e resulted by the three parameters is higher than 1 in 1024 typically used in OC-192 link, and ranged between 1 in 200 to 600.

6.3. Results

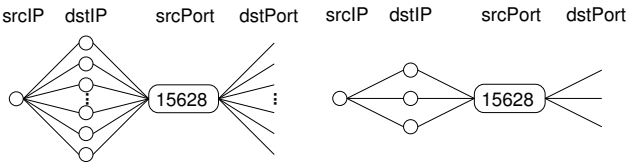
Figure 9(a) studies the impact of sampling on classification accuracy with BLINC on the DORM trace. We define accuracy as $accuracy = c/t$, where c is the total number of correctly classified flows, and t is the total number of flows. The x-axis shows the sampling rate used and the y-axis shows the accuracy of BLINC for the three categories of flows mentioned earlier. The overall accuracy for all flows reduced from around 95% down to 85% as the sampling rate was reduced from 1 to 0.001, which might mislead one to conclude that sampling does not affect classification accuracy. The overall accuracy includes both *std-ports flows* as well as *non-std-ports flows*, on which BLINC performs differently. When only *std-ports flows* were considered, the accuracy was high (almost 100%) across the range of sampling rates as we expected. However, when only *non-std-ports flows* were considered—arguably, the regime where the need for sophisticated classifiers is most critical—the accuracy decreased significantly from 90% over unsampled data all the way down to 30% with a 1 in 1000 sampling.

Figure 9(b) compares the accuracy of BLINC when RelSamp is used compared to when Sampled NetFlow and Flow Sampling are used, focusing mainly on *non-std-ports flows*. To ensure a fair comparison, we require that the packets are sampled at the same effective rate with both schemes. Compared to Sampled Netflow, the benefits of using RelSamp are significant for all sampling rates considered—for instance, for an effective sampling rate of 0.002, the accuracy is 70% with RelSamp and only 30% for Sampled NetFlow.

While RelSamp outperforms Flow Sampling from 0.002 to 0.004, the accuracy gap between RelSamp and Flow Sampling only becomes about 8% after that range. To understand this result better, we investigated how much accuracy Flow Sampling achieved for *std-ports flows*. Interestingly, the classification results were worse than those of RelSamp and Sampled NetFlow (not shown for brevity). Both of them approximately achieve over 97% accuracy, but Flow Sampling only achieves about 80% accuracy. Further, we checked total number of *std-ports flows* as well as *non-std-ports flows* for Sampled NetFlow and Flow Sampling. It turned out that while Sampled NetFlow over-samples *std-ports flows* rather than *non-std-ports flows*, the ratio between two is similar in case of Flow Sampling. This is because Flow Sampling is unbiased in flow size whereas random sampling in NetFlow is biased towards large flows which are found more among *std-ports flows* in our trace. Note that classification in BLINC is triggered by a threshold in number of flows and packets constituting a graphlet. Thus, while the accuracies of both

categories of flows were influenced in case of Flow Sampling because it balances the number of sampled flows in both categories, Sampled NetFlow had worse accuracy in *non-std-ports flows* classification because it less samples those flows.

We also looked at the actual number of correctly classified flows for each sampling schemes. At the right-most data point in Figure 9(c), while RelSamp classifies about twenty eight thousand flows correctly, other two methods successfully classifies only about two thousand flows. On the whole, RelSamp has roughly **10 times** more number of accurately classified flows than Sampled NetFlow and Flow Sampling. Therefore, our RelSamp outperforms the other two sampling methods in terms of accuracy as well as the absolute number of correctly classified flows.



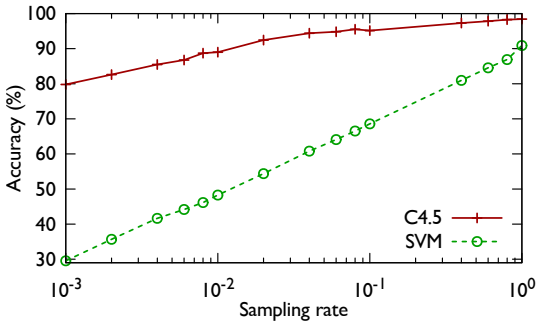
(a) RelSamp (b) Sampled NetFlow

Figure 10: Graphlets of p2p flows by RelSamp and Sampled NetFlow. They have same source IP and port, but different fanout cardinality.

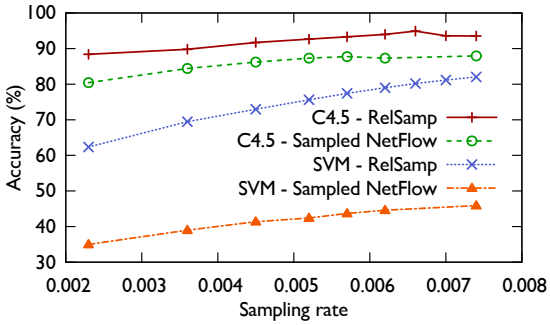
We now discuss the reasons for the significantly improved accuracy with RelSamp. Host-behavior based classifiers such as BLINC work by observing communication patterns of hosts. Sampling affects BLINC by providing distorted hosts' profiles that do not reflect the actual social communication between them. For example, consider a p2p graphlet. The collection of flows matching a graphlet with unsampled data may not match anymore under sampling. Figure 10 shows an actual example from our dataset for the p2p graphlet. Figure 10(a) shows the flows initiated from a certain host as captured by RelSamp, and Figure 10(b) shows flows captured by Sampled Netflow for the same host during the same time period. In this case, RelSamp was able to capture 55 flows which exceeds BLINC's threshold (*fanout* cardinality) for the p2p graphlet. Sampled Netflow, on the other hand, captured only 3 flows which is significantly below the threshold leading to BLINC classifying them as *unknown*.

Figure 11(a) studies the impact of sampling by Sampled Netflow on C4.5 and SVM for *non-std-ports flows*. As sampling rate was lowered, accuracy decreased from 90% down to 30% for SVM, and 98% to 80% for C4.5. We believe the degradation with SVM occurs because the algorithm lacks samples that it needs in training phases; as the number of samples in a training set increases in proportion to sampling probability, the accuracies of SVM increases. In contrast, C4.5 appears relatively more robust to sampling. We hypothesize this may be very likely due to C4.5's entropy-based discretization capability as shown in [23]. That said, a more extensive study using a wide range of traces is required to draw definite conclusions regarding the relative robustness of different classification algorithms. We defer a more detailed investigation of these issues to future work.

Figure 11(b) compares the classification accuracy of C4.5 and



(a) Classification accuracy for *non-std-ports* flows generated by Sampled NetFlow



(b) Comparing classification accuracy for *non-std-ports flows* generated by Sampled NetFlow and RelSamp

Figure 11: C4.5 and SVM classification results.

SVM for *non-std-ports flows* obtained with Sampled Netflow and RelSamp. The figure shows that RelSamp outperformed Netflow by around 10% in C4.5 and 35% with SVM. We believe this is because RelSamp captures more flows than normal NetFlow (even though the number of packets sampled in each scheme is the same), which potentially provides better data for the training phase.

7. Related Work

Due to the importance of measurements in several network management tasks, there exists a lot of prior work [36, 37, 27, 38, 39, 40, 41, 42]) in architecting better sampling-based passive measurement solutions. Despite their existence, to the best of our knowledge, we are the first to introduce this notion of related sampling that can be exploited to make sampled flow records preserve application structure. In this section, we outline some of these solutions and discuss how they differ from our RelSamp architecture.

Several researchers have observed glaring deficiencies in Sampled NetFlow and proposed solutions to address some of them. For instance, Estan *et al.* in [36] propose Adaptive NetFlow to automatically change the sampling rate to better adapt to adversarial traffic patterns such as denial-of-service attacks. Kompella *et al.* propose Flow Slices as a solution to allow better tuning knobs for controlling memory and CPU utilization in [27]. While Flow Slices shares some similarity with RelSamp

in that, both have split stages, the notion of related sampling is completely absent in Flow Slices. Sekar *et al.* have proposed cSamp [42] for network-wide flow monitoring with a goal to minimize redundancies in routers sampling packets independently. RelSamp, on the other hand, is designed to operate within a router.

Beyond sampling frameworks, a few past works have focused on providing network operators with complete flexibility in choosing which flows they want to sample. For example, Yuan *et al.* devised ProgME [38] that provides operators the flexibility to configure hyper-spaces called flowsets, assuming operators know what flows they are interested in. While it offers operators with great flexibility, it is not clear how to exactly define these flowsets to preserve application structure. A similar recent effort is FlexSample [39] by Ramachandran *et al.*, in which they provide a simple language to specify groups of flows of interest using tuples on specific packet header fields, and build counter-based predicates on them. It requires maintaining extra state for approximate checking of the predicates and is designed to provide sophisticated monitoring of traffic subpopulations. RelSamp, on the other hand, does not require any extra state; it is designed to be simple for network operators to implement related sampling mainly geared toward application monitoring. Counter Braids [37] proposed recently by Lu *et al.* proposes a new architecture for sharing counters. Their work is complementary to ours; we can utilize their counter braids in compressing counters to reduce resource usage across flows.

Influence of sampled traffic on anomaly detections has been studied in the past [43, 44]. In [43], Mai *et al.* observed that packet sampling schemes distort original traffic features, and the accuracy of anomaly detection algorithms they tested is impacted by sampling techniques. Our work focused on studying the benefits of RelSamp on traffic classification, but RelSamp can benefit anomaly detection and other attack detection as well, by providing more amount of information to construct anomaly signatures.

Traffic classification is one of the main applications of RelSamp which we have studied in this paper. In general three main approaches emerged—deep packet inspection (DPI) [45, 25, 46], behavior-based [14, 19], and flow-feature based [15, 13, 16, 17, 18]. A recent work [24] demonstrates the possibility of high-speed online classification using a simple Bayes algorithm. However, it is unclear whether the system can sustain when it is equipped with other statistical classifiers such as SVM and C4.5. Thus, while in high speed networks the use of packet sampling is generally inevitable, all previous works have mainly focused on unsampled traffic data. Two of these works [18, 16] have hypothesized that the accuracy of their method will degrade fairly quickly under packet sampling, but neither investigates it further. Our work, in contrast, does not provide a new classification mechanism, but instead provides a sampling scheme that can improve the accuracy of traditional classification techniques.

8. Conclusion

While the wide availability of NetFlow across many modern routers make it an ideal candidate for continuous, low-cost monitoring of network application traffic at enterprise edge routers, the sampling algorithms employed by NetFlow today are inadequate to capture application behavior. In this paper, we have presented RelSamp, an architecture based on the key idea that related flows part of the same application session are sampled with higher probability.

Our evaluations on real traces show the importance and viability of a related sampling approach. RelSamp is capable of obtaining 5-10x more flows per application session compared to Sampled NetFlow and flow sampling. It increases the classification accuracy of BLINC, SVM and C4.5 up to 50% in comparison with the flows output by Sampled NetFlow. We view RelSamp as a first step towards enabling ubiquitous, continuous, low-cost application monitoring. In the future, we hope to explore its applications in detecting undesirable behavior of new applications, and in enabling what-if analysis frameworks that can help network providers reason about the impact of future application trends on the design of their networks.

Acknowledgments

We thank William Harshbarger, Greg Hedrick at Purdue ITaP for their immense help in obtaining the network traces. This work was supported in part by the National Science Foundation through award CNS-0831647, and Cisco Systems.

References

[1] Salesforce: CRM software solutions and enterprise cloud computing, `http://www.salesforce.com`.
[2] Google Apps, `http://www.google.com/apps`.
[3] Evolution of the Web from 2000 to 2007, `http://www.websiteoptimization.com/speed/tweak/evolution-web/`.
[4] H. Xie, Y. Yang, A. Krishnamurthy, Y. Liu, A. Silberschatz, P4P: Provider portal for (P2P) applications, in: ACM SIGCOMM, 2008.
[5] D. Choffnes, F. Bustamante, Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems, in: ACM SIGCOMM, 2008.
[6] Is the Skype outage really a big deal?, `http://news.cnet.com/8301-10784_3-9761673-7.html`.
[7] S. Bellovin, Security Aspects of Napster and Gnutella, invited Talk at USENIX Annual Technical Conference, 2001.
[8] N. Naoumov, K. Ross, Exploiting P2P systems for DDoS attacks, in: International Workshop on Peer-to-Peer Information Management, 2006.
[9] R. Torres, M. Hajjat, S. Rao, M. Mellia, M. Munafò, Inferring undesirable behavior from P2P traffic analysis, in: ACM SIGMETRICS, 2009.
[10] L. MacVittie, The Impact of AJAX on the Network, http://www.ddj.com/web-development/205921196 (2007).
[11] G. Maier, A. Feldmann, V. Paxson, M. Allman, On dominant characteristics of residential broadband internet traffic, in: ACM IMC, 2009.
[12] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, M. Faloutsos, Is P2P dying or just hiding?, in: IEEE Globecom, 2004.
[13] A. Moore, D. Zuev, Internet traffic classification using bayesian analysis techniques, ACM SIGMETRICS Performance Evaluation Review 33 (1) (2005) 50–60.
[14] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: multilevel traffic classification in the dark, in: ACM SIGCOMM, 2005.
[15] A. Mcgregor, M. Hall, P. Lorier, J. Brunskill, Flow Clustering Using Machine Learning Techniques, in: PAM, 2004.
[16] S. Zander, T. Nguyen, G. Armitage, Automated traffic classification and application identification using machine learning, in: Conference on Local Computer Networks, 2005.
[17] Z. Li, R. Yuan, X. Guan, Accurate classification of the internet traffic based on the SVM method, in: IEEE ICC, 2007.

[18] L. Bernaille, R. Teixeira, K. Salamatian, Early application identification, in: ACM CoNEXT, 2006.
[19] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, G. Varghese, Network monitoring using traffic dispersion graphs (tdgs), in: ACM IMC, 2007.
[20] N. Hohn, D. Veitch, Inverting sampled traffic, IEEE/ACM Trans. Netw. 14 (1) (2006) 68–80.
[21] F. Schneider, S. Agarwal, T. Alpcan, A. Feldmann, The New Web: Characterizing AJAX Traffic, in: International Conference on Passive and Active Network Measurement, 2008.
[22] R. Smith, C. Estan, S. Jha, S. Kong, Deflating the big bang: fast and scalable deep packet inspection with extended finite automata, in: ACM SIGCOMM, 2008.
[23] Y.-s. Lim, H.-c. Kim, J. Jeong, C.-k. Kim, T. T. Kwon, Y. Choi, Internet traffic classification demystified: On the sources of the discriminative power, in: ACM CoNEXT, 2010.
[24] P. M. Santiago del Rio, D. Rossi, F. Gringoli, L. Nava, L. Salgarelli, J. Aracil, Wire-speed statistical classification of network traffic on commodity hardware, in: ACM IMC, 2012.
[25] M. Mellia, R. Lo Cigno, F. Neri, Measuring IP and TCP behavior on edge nodes with Tstat, Computer Networks 47 (1) (2005) 1–21.
[26] YAF: Yet Another Flowmeter, `http://tools.netsa.cert.org/yaf/`.
[27] R. R. Kompella, C. Estan, The power of slicing in internet flow measurement, in: IMC, 2005.
[28] YouTube: Broadcast Yourself, `http://www.youtube.com`.
[29] H.-C. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, K. Lee, Internet traffic classification demystified: Myths, caveats, and the best practices, in: ACM CoNEXT, 2008.
[30] K. Bennett, C. Campbell, Support vector machines: hype or hallelujah?, ACM SIGKDD Explorations Newsletter 2 (2) (2000) 1–13.
[31] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, k. claffy, GT: picking up the truth from the ground for Internet traffic, ACM SIGCOMM Computer Communication Review (CCR).
[32] M. Canini, W. Li, A. W. Moore, R. Bolla, Gtvs: Boosting the collection of application traffic ground truth, in: Proceedings of the First International Workshop on Traffic Monitoring and Analysis, 2009.
[33] WEKA: Data-mining Software in Java., `http://www.cs.waikato.ac.nz/ml/weka`.
[34] J. C. Platt, Advances in kernel methods, 1999, Ch. Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pp. 185–208.
[35] N. Williams, S. Zander, G. Armitage, A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification, ACM SIGCOMM CCR 36 (5).
[36] C. Estan, K. Keys, D. Moore, G. Varghese, Building a better netflow, in: ACM SIGCOMM, 2004.
[37] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, A. Kabbani, Counter braids: a novel counter architecture for per-flow measurement, in: ACM SIGMETRICS, 2008.
[38] L. Yuan, C.-N. Chuah, P. Mohapatra, ProgME: towards programmable network measurement, in: ACM SIGCOMM, 2007.
[39] A. Ramachandran, S. Seetharaman, N. Feamster, V. Vazirani, Fast monitoring of traffic subpopulations, in: ACM IMC, 2008.
[40] H. V. Madhyastha, B. Krishnamurthy, A generic language for application-specific flow sampling, ACM SIGCOMM CCR 38 (2) (2008) 5–16.
[41] M. Saxena, R. R. Kompella, A framework for efficient class-based sampling, in: INFOCOM, 2009.
[42] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, D. G. Andersen, cSAMP: a system for network-wide flow monitoring, in: USENIX NSDI, 2008.
[43] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, H. Zang, Is sampled data sufficient for anomaly detection?, in: ACM IMC, 2006.
[44] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, A. Lakhina, Impact of packet sampling on anomaly detection metrics, in: ACM IMC, 2006.
[45] P. Haffner, S. Sen, O. Spatscheck, D. Wang, ACAS: automated construction of application signatures, in: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data, ACM, 2005, p. 202.
[46] T. Choi, C. Kim, S. Yoon, J. Park, B. Lee, H. Kim, H. Chung, T. Jeong, Content-aware Internet application traffic measurement and analysis, in: IEEE/IFIP NOMS 2004.